Building an Effective Software Testing Practice

Date: February 19, 2014

Checkpoint Technologies, Inc.

Email: info@checkpointtech.com
Office: (813) 818-8324
URL: www.checkpointtech.com
# Table of Contents

**Introduction**.................................................................................................................. 4

**How Requirements Impact Testing** .................................................................................. 5

  - Why Requirements Matter ............................................................................................ 5
  - Testing Involvement in Requirements Planning .......................................................... 5
  - The Costs of Correcting Defects .................................................................................. 7

**Unit Testing – Developers Are Testers Too** ................................................................. 8

  - Definition and Benefits of Unit Testing ......................................................................... 8
  - Unit Testing Frameworks ................................................................................................ 8
  - Limitations of Unit Testing ............................................................................................ 8
  - Software Testers and Unit Testing ................................................................................ 9

**System, Regression and End to End Testing** ............................................................... 10

  - The Domain of the Software Testing Practice .............................................................. 10
  - Definition of System Testing ......................................................................................... 10
  - Definition of Regression Testing .................................................................................. 10
  - Definition of End to End Testing .................................................................................. 11

**Test Planning** .................................................................................................................. 12

  - The Role of Documentation and Standards in Test Planning ......................................... 12
  - The Test Plan ................................................................................................................ 12
  - Test Cases .................................................................................................................... 13
  - Requirements Traceability ............................................................................................. 13
  - Defect Management Process ......................................................................................... 14

**Manual Testing and Automated Testing** ..................................................................... 15

  - The Manual Testing Skillset ......................................................................................... 15
  - Manual Testers and Exploratory Testing ...................................................................... 15
  - Automated Testing ....................................................................................................... 16
  - To Automate or Not to Automate .................................................................................. 16
  - An Automation Proof of Concept .................................................................................. 17
  - The Strengths of Test Automation ................................................................................ 17
  - Realizing Success and ROI With Test Automation ...................................................... 17
Performance, Load and Stress Testing..................................................19
  Definitions ....................................................................................... 19
  How These Types of Testing are Accomplished ......................... 19
  Why Performance Test?................................................................. 19
  The Performance Testing Skillset .................................................. 20
  Considerations For Performance Testing ................................. 20

Security Testing..................................................................................21
  Definition of Security Testing ......................................................... 21
  Why Test Security? ......................................................................... 21
  Profile of a Security Tester .............................................................. 21
  Security Test Automation ................................................................. 21

User Acceptance and Beta Testing.....................................................22
  Definitions ....................................................................................... 22
  Benefits of User Acceptance Testing ............................................. 22
  How the Software Test Team Can Aid in UAT ............................. 22

Summary ............................................................................................. 23

About Checkpoint Technologies .........................................................24
Introduction

All organizations involved in software development have experienced the frustration of having defects reported by production users. The severity of software “bugs” ranges from those that merely inconvenience the user community to critical flaws that cause major disruption, loss of revenue and damaged credibility. Developers scramble to introduce patches and post-mortem analysis is conducted to determine how the problems escaped the various phases of software testing.

Building an effective software testing practice is difficult, and many organizations slowly construct and modify their testing processes based on the lessons learned through repeated failure. Worse still, some software development shops run into the same issues time and time again, but due to aggressive schedules and lack of resources, never introduce the change necessary to make the testing process more efficient and effective. When present, there can be many other reasons for this lack of change including: budget constraints, time to market considerations and lack of recognition for the value of software testing from the business.

This white paper outlines the role of testing in the software development lifecycle and focuses specifically on the test team activities and types of testing that can be implemented to improve software quality. Each of the testing roles and disciplines mentioned herein has strengths and blind spots. Recognizing the types of issues each is designed to uncover, and the types of issues each will probably not find, is a large part of building an effective software testing practice.

This paper was written for organizations both large and small and is not tied to a specific software development methodology. Whether you use a full waterfall method or take an agile approach, the same basic concepts apply. The information contained herein is based on industry standards and many years of software quality consulting, observing what worked and what didn’t, and highlighting the common denominators of all successful software testing groups.
How Requirements Impact Testing

Why Requirements Matter
Regardless of your software development approach, clear, unambiguous and testable requirements are the key to successful software testing. Requirements are documented in many different forms depending on the organization. They may be presented in a traditional requirements specification, use cases written by business analysts or user stories created in an agile setting. These requirements form the basis for and are distilled into technical documentation like interface specifications, design documents, data dictionaries, etc. All of this documentation determines the content of the test plan and test cases. For the purposes of this section, all of these artifacts will be referred to using the general term “requirements.”

This paper will not go into detail on the process of writing testable requirements; this is an art in itself and the subject of many well-known textbooks and standards documents. The various software development methodologies approach requirements management differently. Traditional waterfall approaches enforce “requirement freezes” and a change control process, while agile methodologies focus on smaller units of code and are designed to be more adaptable to requirements change during development. Regardless of how requirements are managed, testing will not succeed unless the testers are aware of how the application needs to behave.

Without clear requirements, experienced system testers may still find obvious defects, e.g.: when a certain menu option is selected or button pressed, the software application shuts down with an error. Without clear requirements, design documentation will suffer and testers may not be able to ferret out issues in application behavior that will quickly be detected by business users, e.g.: when creating an invoice, the sales tax multiplier is based on the customer’s state of residence.

Testing Involvement in Requirements Planning
The following key points apply to software testing involvement in requirements planning:

- Requirements are developed through interaction with the customer, first in requirements definition meetings and later in Joint Application Design (JAD) sessions. Testing needs to be represented in this process even if in a passive role. This ensures that testers understand the customer’s software needs just as well as the architects and developers.

- Through participation during the planning phase of a software project testers can foster a relationship with the customer and work with business analysts to help refine requirements as they are written. Both of these relationships can be just as
critical as a good relationship with developers to assist in finding germane defects during system testing.

- Testers should always apply a testing focused lens to requirements, asking the questions “Can this requirement be tested? If so, how will I test it? Are there special data or environmental considerations?”

- The testing organization should review and sign off on all technical documentation produced by the development team and analysts. These documents are a result of requirements analysis and determine what will actually be constructed. Testers that participate in the planning process are often in a good position to point out contradictions between requirements and technical documentation. Review of this material also helps solidify the approach to testing the design.

All of the above points highlight an important concept: well written requirements are a precursor to effective software testing. Testers cannot do it alone, successful software testing requires the testing group to reach out to other groups in the organization such as the customer, business analysts and developers. This will help to ensure they have everything they need to write test cases and find critical defects before the application is released to production.
The Costs of Correcting Defects

Industry data clearly shows, as illustrated in figure 1 below, the costs of fixing software defects will increase significantly with each phase in the software development process. The requirements phase is the most cost effective point in the process to discover and remediate issues that if left untouched will result in an application that doesn’t meet business needs.

![Costs of Correcting Defects](image)

**Figure 1 - Cost of Correcting Defects By Lifecycle Phase**

The above figure is an excellent way to demonstrate the value of software testing and encourage investment from the business. Many thousands of dollars are saved when defects are detected before the maintenance phase (production). The average cost has also been shown to rise based on the number of defects found.
Unit Testing – Developers Are Testers Too

Definition and Benefits of Unit Testing

In computer programming, a unit test is a method of testing the correctness of a particular module of source code. Unit testing is performed during the development and coding process and developers generally execute unit tests on source code they have written. As seen in figure 1 above, the cost of finding defects increases over seven times between coding and testing, therefore unit testing can provide a huge return on investment when it is implemented effectively.

By targeting specific code modules, unit tests allow developers to find defects in an isolated piece of code. Defects found during unit testing are generally not entered into a defect tracking system; instead the developer executing the test documents and repairs them straightaway.

Unit testing also helps developers understand code dependencies. A benefit of this is that unit testing often motivates developers to create discreet and decoupled code modules.

Unit Testing Frameworks

Because a unit test executes code outside of an integrated environment, test frameworks must be used to exercise the programmatic inputs and outputs. Many Integrated Development Environments (IDEs) include automated tools to build these frameworks around a code module. For languages such as JAVA, open source tools are available to help in unit test creation and execution.

Many unit test framework tools are self-documenting in terms of the test cases and results. Developers can add their own test criteria to the suggested unit test cases and run them as often as required.

Limitations of Unit Testing

Because unit test cases are run inside a framework, and not in the integrated environment in which the code will ultimately exist, there are limitations to the types of issues detected. Unit test cases are only as good as their test criteria. A developer may not be aware of all cases of input the unit will receive when the code is integrated. Unit test cases will also not detect issues caused by other units of code, integration or performance issues.

---

Software Testers and Unit Testing

While unit testing is a development exercise, it is critical for a software testing organization to stress the importance of unit testing to development counterparts.

Some of the ways a testing group can encourage and leverage unit testing are:

- Participate in reviews of unit test cases. These sessions will encourage developer documentation and can also serve to educate system testers as to how code will actually operate. Testers can also use these sessions for informal risk analysis, asking questions such as “What would be the result of this unit failing in production?”

- Require unit test results as an entrance criterion for system testing. This will ensure that unit test cases were executed and also give system testers insight into problematic or complex areas of code.

- Assist developers in creation of test data or other tasks which will further unit test execution. Often the system test team can use their testing expertise to aid developers in creating effective test cases, which benefits everyone.
System, Regression and End to End Testing

The Domain of the Software Testing Practice

While the activities detailed in the two previous sections (requirements planning, unit testing) are key to an organization’s overall software quality, parties other than the software test team generally control them. In system, regression and “end to end” testing we are entering phases of the software lifecycle that are the domain of the software testing practice. The software testing team within an organization will plan and execute these phases of software testing.

These areas (along with performance testing and security testing, discussed later) are the most common responsibilities of the software testing team. The results of these phases of testing will determine the perceived effectiveness of the test team and establish the test team’s credibility within a company.

Definition of System Testing

System testing is “testing that attempts to discover defects that are properties of the entire system rather than of its individual components.”¹² This is the phase where an application build is assembled or compiled by development and delivered to the test team. System test cases are then executed against the build.

System testing occurs after development has completed unit testing. Defects found in system testing are documented in a defect-tracking system and evaluated by the development team. If the defect report satisfies the criteria of a true defect necessitating an immediate fix, it will be repaired and delivered in a subsequent build. The test team will then re-test the application to verify the fix. After completion of system and regression testing, a build will be designated as release ready, and prepared for performance and user acceptance testing (discussed later).

NOTE: Different organizations and industry groups use different terms for test phases and phases may vary slightly. Some have phases called “integration testing” or “functional testing” that correspond to what this paper is referring to as system testing, while some use the term “integration testing” to refer to what this paper defines “end to end” testing.

Definition of Regression Testing

Regression testing is “retesting of a previously tested program following modification to ensure that faults have not been introduced or uncovered as a result of the changes

¹² Site Test Center, “Software Testing Glossary”, sitetestcenter.com
Regression testing is generally executed in parallel with system testing. Having a comprehensive regression test suite is critical to determining if a software product is fit for release. All of the major functions of a system should be covered, and determinations as to which regression tests need to be run can be made as software testing efforts progress. Nothing will undermine confidence in a software testing practice like a released product containing “broken” features that worked in the previous release. Users will sometimes accept a certain number of defects in new functionality, when pre-existing functionality is compromised, credibility will suffer.

Regression Testing is one of the strong suits of software test automation (discussed below) and with automation tools regression coverage can be increased dramatically. Regression test automation also frees manual testers to concentrate on system testing new functionality.

**Definition of End to End Testing**

End to end (E2E) testing is the testing of a full business process, including interaction between systems via interfaces. Most organizations rely on multiple information systems to process a business transaction. A customer may interact with a website which interacts directly or indirectly with a billing system, an inventory control system, a marketing system, etc. With such complex system architecture and interfacing between systems, testing a full business process when change is introduced serves an important role in an organization’s software quality methodology.

Often an E2E test will involve multiple testing groups within an organization, and an E2E test can include developers and end users in areas where formalized system testing does not exist. Because of the resources involved, coverage is generally limited to a positive path test of a business process, with verification points throughout the process.

As mentioned earlier, some organizations refer to E2E testing as “integration testing” whereas others use the term “integration testing” to mean testing integration of the components of a single application. Using the term E2E avoids this confusion.

---

Test Planning

The Role of Documentation and Standards in Test Planning

Documentation is key in any test planning effort. Even in agile methodologies, where code and product are emphasized over documentation, a testing practice must have repeatable, documented test procedures in order to succeed. Efficient documentation and repeatability requires standardization of major testing deliverables and processes.

Some of the testing documents and processes that should be standardized include:

- The Test Plan
- Test Cases
- Document and deliverable naming conventions (e.g. test cases, defect reports, etc.)
- Automation scripts (coding and documentation standards)
- The Defect Management Process
- Requirements Traceability
- Status and Progress Reports
- Test Completion Reports

The Test Plan

A successful software testing practice requires thoughtful and standardized test planning and involves multiple deliverables. The key software testing deliverable is the test plan. Test plans come in many flavors and there are several industry standard templates available. A test plan would include but not be limited to the following information:

- Software to be tested
- Objective of testing effort
- Responsibilities and contact info
- Testing risks
- Testing dependencies
- Testing constraints
- Scope and limitations of testing effort
- System testing strategy
- Regression testing strategy
- Testing schedule
- Test entrance and exit criteria
- Test environment identification
- Test data requirements
- Test automation strategy
- Testing tools to be used
• Test script management strategy
• Requirements traceability
• Metrics and reports to be generated
• Defect management strategy

Most test plan templates include sections for all of the above information and more. Some of the information in a test plan will be boilerplate, while other sections will require rework based on the project. Some data for the test plan may be taken from other documents or automated repositories. The test plan may also contain references to these documents and repositories.

A test plan should be reviewed and signed off on by other groups within the project team including the development team, business analysts, customer liaisons and sometimes the customer themselves. Just as testers can provide valuable insight into requirements and design, other groups can give valuable input into your test strategy.

Test Cases
Test Cases detail the actual test steps and expected results that will be used during the testing effort. Test cases can be documented in office documents or stored in automated test repositories. Test case documentation should be standardized and test cases need to be repeatable to be valid. Some of the information included in test case documentation includes:

• Test Case Name (standardized)
• Test Case Author
• Date Created
• Test case objective
• Application process being tested
• Test Case Steps and Expected Results
• Entry and Exit Conditions
• Data Requirements
• Changes to test case

Automated repositories provide a mechanism for storing test cases and associated info and generating documentation. They also may include features allowing users to execute tests, document test results and trace test cases to requirements and defects.

Requirements Traceability
Tracing test cases to requirements helps ensure you are providing adequate test coverage. Generally requirements traceability to test cases is part of a larger traceability effort. Developers will also be tracing code modules to requirements. Traceability is often
documented using a Requirements Traceability Matrix, which the test team contributes to but does not own. Requirements may also be stored in a requirements repository tool and traceability achieved through integration with a test case management tool. Automated traceability has many benefits, including instant recognition and notification of changes to requirements and test cases.

**Defect Management Process**
The process of managing defects within an organization involves multiple groups actively and passively participating in the process. A defect lifecycle that outlines the various statuses and roles involved in the process should be developed and standardized. An automated tool may be used to track defects, allowing for much more efficient defect management, role based security, auditing, automatic notification and automated reporting.
Manual Testing and Automated Testing

The Manual Testing Skillset

The manual tester is the cornerstone of an effective software testing practice. Manual testing involves a varied and complex skill set. While manual testers are often drawn from the end user community and bring extensive application knowledge, effective manual testing involves much more than just comprehension of how an application is used in production.

In fact, a skilled manual tester brings a set of capabilities that allows them to effectively test any application. Specific business knowledge must still be obtained, but once acquired it is integrated into the manual tester’s understanding of testing techniques and application architecture and behavior.

According to James A. Whitaker, in his article “What Is Software Testing? And Why Is It So Hard?” the following applies to manual testers:

“To plan and execute tests, software testers must consider the software and the function it computes, the inputs and how they can be combined, and the environment in which the software will eventually operate. This difficult, time-consuming process requires technical sophistication and proper planning. Testers must not only have good development skills—testing often requires a great deal of coding—but also be knowledgeable in formal languages, graph theory, and algorithms. Indeed, creative testers have brought many related computing disciplines to bear on testing problems, often with impressive results.”

Manual testing is a cognitive process. Manual testers leverage their cumulative knowledge stores and the advanced pattern recognition mechanism of human intelligence during test execution. An effective manual tester will detect not only differences between actual and expected results, but also discover related and unrelated defects which may manifest themselves in a test run.

Manual Testers and Exploratory Testing

In his article “What is Exploratory Testing?” Industry expert James Bach gives the following explanation of exploratory testing:

---

“The plainest definition of exploratory testing is test design and test execution at the same time. This is the opposite of scripted testing (predefined test procedures, whether manual or automated). Exploratory tests, unlike scripted tests, are not defined in advance and carried out precisely according to plan.”

Bach goes on to illustrate the difference between exploratory testing and scripted testing:

“The scripted approach to testing attempts to mechanize the test process by taking test ideas out of a test designer's head and putting them on paper. There's a lot of value in that way of testing. But exploratory testers take the view that writing down test scripts and following them tends to disrupt the intellectual processes that make testers able to find important problems quickly.”

Exploratory testing does have guidelines and does require documentation for repeatability, two attributes that differentiate it from unstructured “ad-hoc” testing. Exploratory testing can be used as a compliment to scripted testing.

**Automated Testing**

Automated testing involves using a software test automation tool to create scripts that perform test actions and record and compare actual results. Test automation is a powerful option in the software testing war chest and depending on the tool used can provide many benefits. That being said, there are many considerations that must be taken into account before test automation is attempted.

**To Automate or Not to Automate**

Automation can be a daunting prospect for a software testing organization. The investment can be substantial and for a return on investment to be realized, careful planning needs to occur.

Factors that need to be considered before attempting test automation include:

- What are our automation goals?
- What tool(s) best suit our automation needs?
- Who will do it?
- What kind of hardware will automation require?
- Do I need a new test environment for automation?
- What skills and training may be required to effectively leverage automation?

---

6 ibid
• What types of test cases do we plan to automate?
• How can we design our automation to be maintainable?
• Should we hire test automation consultants to help?

An Automation Proof of Concept
Before diving into test automation, an organization may want to try an automation proof of concept. Regression tests that are repeated for every build of an application are a good place to start. If the proof of concept is successful, the benefits of test automation will be obvious.

The Strengths of Test Automation
Test Automation can help a software testing organization in a myriad of ways including:

• Automated Regression Testing – tests run on every build of an application are excellent candidates for automation.
• Increased Test Coverage – test automation allows for a level of test coverage that is unattainable using manual testers alone.
• Data Driven Testing - automation tools allow testers to create a single test script and then magnify test coverage through data substitution. Essentially the test script executes in a loop. As it iterates, data variations are entered. This can be useful in both system and regression testing.
• Unattended Test Execution – Automated scripts can be run at any time of night or day if the target application is available. This frees up manual resources and increases test coverage.
• Multiple Levels of Verification – Automated scripts can be designed to enter values into a GUI and verify both the GUI and the underlying database.
• Non-GUI testing - It’s almost always easier to automate command line and API tests than it is to automate GUIs.
• Test Environment and Test Bed Setup – Repetitive setup tasks can be automated.
• Data Creation – Automated test tools can be used to create volumes of test data.

Realizing Success and Return on Investment With Test Automation
There are many things an organization can do to ensure success and return on investment with test automation. These include:

• Proper Planning – in order to automate test cases, test automation goals should be documented and automation friendly test cases selected. Having documented test procedures and expected results will vastly streamline automation efforts.
• Managing Expectations – test automation requires an investment of both time and resources. After investing lots of capital in test automation, many organizations expect immediate results from test automation. While some short-term gains can be realized by leveraging data driven capabilities, the majority of the automation effort will not produce returns until the scripts have been able to mature through multiple software releases.

• You Can’t Automate Everything – This is related to the two points above. Test automation must be recognized as a compliment for manual testing which allows for greater test coverage. It must be stressed that it does not allow for the elimination of all manual testing.

• Apply Software Development Best Practices to Automated Test Scripts – Ensure that there are scripting standards, documentation standards and configuration management processes in place for automated test scripts.

• Dedicate Automation Resources – One of the main reasons automation efforts fail is because manual software testers are drafted into “part time automation” roles. To do justice to test automation, resources must be dedicated. Part of the return on investment involves testers effectively using and growing more skilled with the automation tools. The more a tester knows about the automation tool, the more sophisticated the scripts that result.

• Pursue Training – Automated test tools are complex and powerful and in order to get the return on investment a testing organization needs to train its automation resources.

• Hire Experts to Help – It is often helpful to hire experts in software test automation to help get your organization on the road to automation success. Consultants can be helpful in targeting test automation and helping you to leverage all of its capabilities. Some companies also benefit from augmenting staff with resources skilled in test automation, rather than focusing efforts on retraining existing resources.

• Design for Maintainability – Automated test scripts require regular maintenance. Time spent designing test scripts with this in mind can help sustain momentum in your test automation effort. Most test automation tools allow for creation of functions and function libraries. Employing functions and code modularity can lower the amount of maintenance required extensively.

• Champion the Results of Automation – Organizations need to share the benefits of automation with all. Reports and graphs demonstrating increased test coverage and defects detected by automation can help continue to justify the investment.
Performance, Load and Stress Testing

**Definitions**
Performance testing is conducted to evaluate the compliance of a system or component with specified performance requirements. Performance requirements will often be defined as part of business requirements during the requirements definition phase of system development. These generally include usage predictions on which scenarios can be modeled.

Load testing is the process of running a number of clients simultaneously to test a client-server system and measure response times. Performance is measured under load, hence load testing is performance testing.

When the load placed on the system is raised beyond normal usage patterns, in order to test the system's response at unusually high or peak loads, it is known as Stress Testing.

Other terms used for this type of testing are “Volume Testing” and “Reliability Testing”.

**How These Types of Testing are Accomplished**
Performance, load and stress testing are accomplished using automated tools which simulate multiple clients connecting to an application. Advanced versions of these tools usually provide for the following capabilities:

- Scripting of application business processes
- Support for native protocols (e.g. HTTP)
- Scaling of system load using “virtual users”
- Creation of Scenarios to specify and schedule virtual user load, ramp up and test duration
- Parameter substitution to vary data and fully exercise system
- Performance monitors for individual system components (e.g. Web Server, Database)
- Monitors of system hardware performance
- Detailed reporting of Performance Metrics

**Why Performance Test?**
Performance testing provides many benefits, including the ability to:

- Accurately determine system throughput and capacity
• Simulate thousands of concurrent users
• Identify system breaking points, weak links, and bottlenecks
• Find applications that do not scale well
• Quantify application or server performance with realistic workloads
• Identify defects which only manifest themselves under load

The Performance Testing Skillset
Performance testing is one of the most technically challenging roles in software testing. A skilled performance tester needs to have an understanding of the performance testing tool, system development, system architecture, network architecture and protocols, databases and hardware capacity. A performance tester will also need a good set of interpersonal skills as they need to interact with many different resources within the organization including business analysts, system administrators, network administrators, developers and database administrators.

Considerations For Performance Testing
There are many considerations for a testing organization pursuing performance testing. These range from infrastructure concerns to performance requirement analysis. A list of some of the questions that should be answered before pursuing performance testing:

• Do we have documented performance requirements? Are they realistic?
• Can we model current production usage to aid in designing performance tests?
• What is the best performance testing tool to meet our performance testing goals?
• What protocols must be supported to load test our system?
• What hardware is required to generate load and measure performance?
• Do I need an isolated network segment to performance test?
• Do I need to build a new test environment for performance testing?
• Are the different skillsets required (System Admins, DBAs, Network Admins, Developers, etc.) within the organization available to aid in performance testing and analysis
• Who is going to perform the testing?
• Should I hire a consultant or a new employee or train an existing resource?
• Would it be better to performance test my application externally using a vendor?
Security Testing

Definition of Security Testing

Security testing verifies that an application provides access only to authorized users of the system. Security testing incorporates functional testing (e.g. password strength) and penetration testing.

Wikipedia defines penetration testing as “…a method of evaluating the security of a computer system or network by simulating an attack by a malicious hacker. The process involves an active analysis of the system for any weaknesses, technical flaws or vulnerabilities. This analysis is carried out from the position of a potential attacker, and can involve active exploitation of security vulnerabilities.”

Why Test Security?

The importance of security testing cannot be stressed enough due to highly distributed system architecture and the Internet. Along with monetary loss, compromised security and unauthorized access to sensitive data can expose an organization to industrial espionage, loss of customer confidence, litigation and criminal prosecution.

Several high profile security breaches have been documented in the media, perpetrated by individuals as varied as teenage hackers, identity theft rings and terrorists.

Profile of a Security Tester

A good security tester will be someone who is well versed in the vulnerabilities of your software platform. Continuing education in this area is a must, as vulnerabilities are constantly being discovered in existing software and introduced in new software.

Security testers interact with many different resources within an organization to ensure that the latest security patches have been applied and adequate measures have been taken in all vulnerable areas of the system architecture.

Security Test Automation

A recent development in the world of security automation is the introduction of functional test focused tools that attempt to exploit known software vulnerability profiles.

---

User Acceptance and Beta Testing

Definitions
The final topic covered in this white paper concerns testing activities performed by end users. User acceptance testing (UAT) is usually the final phase of testing before an application is released. End users exercise the application in a test environment with a predefined set of test cases to verify the application meets acceptance criteria. Often end users will create a UAT test plan and test cases. Results of these tests will be documented and defects opened if applicable.

Beta testing involves releasing the application into production to a select number of production users. The application is then used in a real production environment to verify application behavior. Beta testing is sometimes not possible depending on what components of an application are being released. If the “new” release cannot co-exist with the existing production release, you will not be able to perform beta testing.

Benefits of User Acceptance Testing
End users bring a testing perspective that is valuable in that they are using the application as it will be used in production. Often user acceptance testing will expose defects not found in system testing because system testers were not aware of specific production scenarios.

How the Software Test Team Can Aid in User Acceptance Testing
Software testing teams can do much to promote UAT and champion the value of a UAT testing phase. UAT can do much to ensure a smooth software rollout, and prevent release day “surprises” that occur when software the users have never seen before is promoted to production.

During UAT planning, the system test team can aid the UAT testers by helping them prepare a test plan and helping them document their test cases. While production users have extensive system knowledge, they usually lack testing skills. Aiding in these areas can help foster relationships with end users that can be invaluable to a software testing organization.
Summary

This white paper has covered most of the areas of the software development lifecycle and how they relate to building an effective software testing practice. Hopefully readers will take away concepts they can apply in their own software testing practices, whether they are fine-tuning an existing practice or starting from scratch.

Some of the major points to remember include:

- Get testing involved at the very beginning of a software project
- The earlier in the lifecycle a defect is detected, the cheaper it is to fix
- Unambiguous and testable requirements are critical to software testing
- Encourage and participate in unit testing
- System and regression testing encompass the major responsibilities of most software testing practices
- End to End testing can help identify issues caused in a multi-system business process
- Test Planning Documentation and Deliverables include:
  - Testing Standards
  - Test Plans
  - Test Cases
  - Requirements Traceability
  - Defect Management
- Manual testers are the cornerstone of a software testing practice
- Automated testing can provide many benefits and also has many considerations
- Performance testing can validate performance requirements, find bottlenecks and identify issues seen only under load
- Security testing identifies system vulnerabilities and protects organizations from exposure to liability
- User Acceptance and Beta testing help to ensure end user satisfaction with released software
About Checkpoint Technologies

Checkpoint Technologies, Inc. is a solutions provider with unparalleled expertise in the areas of Quality Assurance and software testing. Incorporated in January 2003, we specialize in all areas of Quality Assurance and software testing (with a focus in functional testing, application security testing, mobile application testing and performance testing). Our best-of-breed solutions and expert services enable you to deliver higher quality software applications to market faster! We provide staff augmentation, consulting services, training, mentoring, and software product sales and support throughout North America.

Checkpoint Technologies is an HP Software Gold Partner, HP Software Support Partner, HP ESSN Partner, and HP Authorized Training Partner. In addition, we also are proud to be a partner with and offer solutions developed by Turnkey Solutions, Perfecto Mobile, Mobile Labs, and Shunra.

Checkpoint Technologies is Software Quality. Assured!

Please visit us at www.checkpointtech.com.

Feel free to contact us today at:
Email: info@checkpointtech.com
Phone: (813) 818-8324